International Telecommunication Union

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

F.751.2

(08/2020)

SERIES F: NON-TELEPHONE TELECOMMUNICATION SERVICES

Multimedia services

Reference framework for distributed ledger technologies

Recommendation ITU-T F.751.2

International Telecommunication Union

ITU-T F-SERIES RECOMMENDATIONS

NON-TELEPHONE TELECOMMUNICATION SERVICES

TELEGRAPH SERVICE	
Operating methods for the international public telegram service	F.1–F.19
The gentex network	F.20–F.29
Message switching	F.30–F.39
The international telemessage service	F.40–F.58
The international telex service	F.59–F.89
Statistics and publications on international telegraph services	F.90–F.99
Scheduled and leased communication services	F.100–F.104
Phototelegraph service	F.105–F.109
MOBILE SERVICE	
Mobile services and multidestination satellite services	F.110–F.159
TELEMATIC SERVICES	
Public facsimile service	F.160–F.199
Teletex service	F.200–F.299
Videotex service	F.300–F.349
General provisions for telematic services	F.350–F.399
MESSAGE HANDLING SERVICES	F.400–F.499
DIRECTORY SERVICES	F.500–F.549
DOCUMENT COMMUNICATION	
Document communication	F.550–F.579
Programming communication interfaces	F.580–F.599
DATA TRANSMISSION SERVICES	F.600–F.699
MULTIMEDIA SERVICES	**F.700–F.799**
ISDN SERVICES	F.800–F.849
UNIVERSAL PERSONAL TELECOMMUNICATION	F.850–F.899
ACCESSIBILITY AND HUMAN FACTORS	F.900–F.999

For further details, please refer to the list of ITU-T Recommendations.

Recommendation ITU-T F.751.2

Reference framework for distributed ledger technologies

Summary

Recommendation ITU-T F.751.2 specifies the reference architecture for distributed ledger technology (DLT), its hierarchical relationship and specific functions, important modules and specific functions in its structure, as well as the main technical route and direction of its core module. Recommendation ITU-T F.751.2 can be used as a guideline for DLT service providers to build systems and for organizations to select and use a DLT platform.

History

Edition	Recommendation	Approval	Study Group	Unique ID*
1.0	ITU-T F.751.2	2020-08-13	16	11.1002/1000/14334

Keywords

Architecture, blockchain, components, DLT, distributed ledger technology, functions, ledger, platforms, reference architecture.

* To access the Recommendation, type the URL http://handle.itu.int/ in the address field of your web browser, followed by the Recommendation's unique ID. For example, http://handle.itu.int/11.1002/1000/11830-en.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications, information and communication technologies (ICTs). The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

Compliance with this Recommendation is voluntary. However, the Recommendation may contain certain mandatory provisions (to ensure, e.g., interoperability or applicability) and compliance with the Recommendation is achieved when all of these mandatory provisions are met. The words "shall" or some other obligatory language such as "must" and the negative equivalents are used to express requirements. The use of such words does not suggest that compliance with the Recommendation is required of any party.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had not received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementers are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database at http://www.itu.int/ITU-T/ipr/.

Table of Contents

Recommendation ITU-T F.751.2

Reference framework for distributed ledger technologies

1 Scope

This Recommendation specifies the reference architecture for distributed ledger technology (DLT), its hierarchical relationship and specific functions, important modules and specific functions in its structure, as well as the main technical route and direction of its core module.

This Recommendation can be used as a guideline for DLT service providers to build systems and for organizations to select and use a DLT platform.

2 References

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published. The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

None.

3 Definitions

3.1 Terms defined elsewhere

This Recommendation uses the following terms defined elsewhere:

3.1.1 asset [b-ITU-T F.751.0]: A representation of value. It can be a diamond, a unit of currency, items inside a shipping container, etc. An asset can be physical or virtual.

3.1.2 blockchain [b-ITU-T F.751.0]: A type of distributed ledger that is composed of digitally recorded data arranged as a successively growing chain of blocks with each block cryptographically linked and hardened against tampering and revision.

3.1.3 consensus [b-ITU-T F.751.0]: Agreement that a set of transactions is valid.

3.1.4 entity [b-ITU-T F.751.0]: Anything that has a separately identifiable existence (e.g., organization, person, group, smart contract or device). An entity uses distributed ledger technology to solve the problem of its business or information systems.

3.1.5 entity address [b-ITU-T F.751.0]: Identifier for one or more entities performing transactions or other actions in a blockchain or distributed ledger network.

3.1.6 ledger [b-ITU-T F.751.0]: Information store that keeps final and definitive (immutable) records of transactions.

3.1.7 node [b-ITU-T F.751.0]: Device or process that participates in a distributed ledger network.

3.1.8 permissionless distributed ledger system [b-ITU-T F.751.0]: Distributed ledger system where permissions are not required to maintain and operate a node.

3.1.9 public DLT; public distributed ledger system; public distributed ledger technology system [b-ISO 22739]: A public DLT has transaction records that are readable by anyone.

3.1.10 private distributed ledger technology system [b-ISO 22739]: DLT system that is accessible for use only to a limited group of DLT users.

3.1.11 smart contract [b-ITU-T F.751.0]: Program written on the distributed ledger system that encodes the rules for specific types of distributed ledger system transactions in a way that can be validated, and triggered by specific conditions.

3.1.12 token [b-ITU-T F.751.0]: A digital representation of value on a shared distributed ledger that is owned and secured using cryptography to ensure its authenticity and prevent modification or tampering without the owner's consent.

3.1.13 transaction [b-ITU-T F.751.0]: An incident or an operation that leads to a change in the status of a ledger, such as adding a record or equivalent exchange based on currency.

3.2 Terms defined in this Recommendation

This Recommendation defines the following terms:

3.2.1 balance model: Model that keeps track of the balance of each account as a global state. The balance of an account is checked to make sure it is greater than or equal to the spending transaction amount.

NOTE – Adapted from [b-ITU-T TS FG DLT D3.1].

3.2.2 event model: Model where consensus takes place upon event (transaction).

NOTE – Originally published in [b-ITU-T TS FG DLT D3.1].

3.2.3 state model: Model where consensus takes place upon state (result).

NOTE – Originally published in [b-ITU-T TS FG DLT D3.1].

3.2.4 unspent transaction output (**UTXO) model**: Model where transaction spends output from prior unspent transactions and generates new outputs that can be spent in the future. The unspent transactions are kept in each node.

NOTE – Adapted from [b-ITU-T TS FG DLT D3.1].

3.2.5 trust endorsement: An endorsement for entities inside DLT systems to trust each other, e.g., for one node to trust other nodes. Instances can be a programmable governance model, or contracts or agreements signed by node owners.

4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

AAA Authentication, Authorization and Accounting

API Application Programming Interface

BFT Byzantine Fault Tolerance

CA Certificate Authority

DApp Decentralized Application

dBFT Delegated Byzantine Fault Tolerance

DID Decentralized Identity

DLT Distributed Ledger Technology

DPoS Delegated Proof of Stake

EVM Ethereum Virtual Machine

IBE Identity-Based Encryption

JVM Java Virtual Machine

P2P Peer to Peer

PKI Public Key Infrastructure

PoS Proof of Stake

PoW Proof of Work

RPC Remote Procedure Call

SDK Software Development Kit

TEE Trust Execution Environment

TXO Transaction Output

UTXO Unspent Transaction Output

VBFT Byzantine Fault Tolerance with Verifiable Randomness

VM Virtual Machine

5 Conventions

In this Recommendation:

– The keywords "is required to" indicate a requirement which must be strictly followed and from which no deviation is permitted if conformance to this Recommendation is to be claimed.

– The keywords "is recommended" indicate a requirement which is recommended but which is not absolutely required. Thus, this requirement need not be present to claim conformance.

– The keywords "can optionally" indicates an optional requirement which is permissible, without implying any sense of being recommended. This term is not intended to imply that the vendor's implementation must provide the option and the feature can be optionally enabled by the network operator/service provider. Rather, it means the vendor may optionally provide the feature and still claim conformance with this Recommendation.

6 Architecture overview

6.1 Overview

The high-level architecture constrains the highly abstract hierarchy of distributed ledgers. The high-level architecture can cover almost all distributed ledgers, including public chains represented by Ethereum [b-ethe] and Bitcoin [b-bitc], private chains represented by Hyperledger Fabric and non-blockchain distributed ledgers systems. See Figure 1.

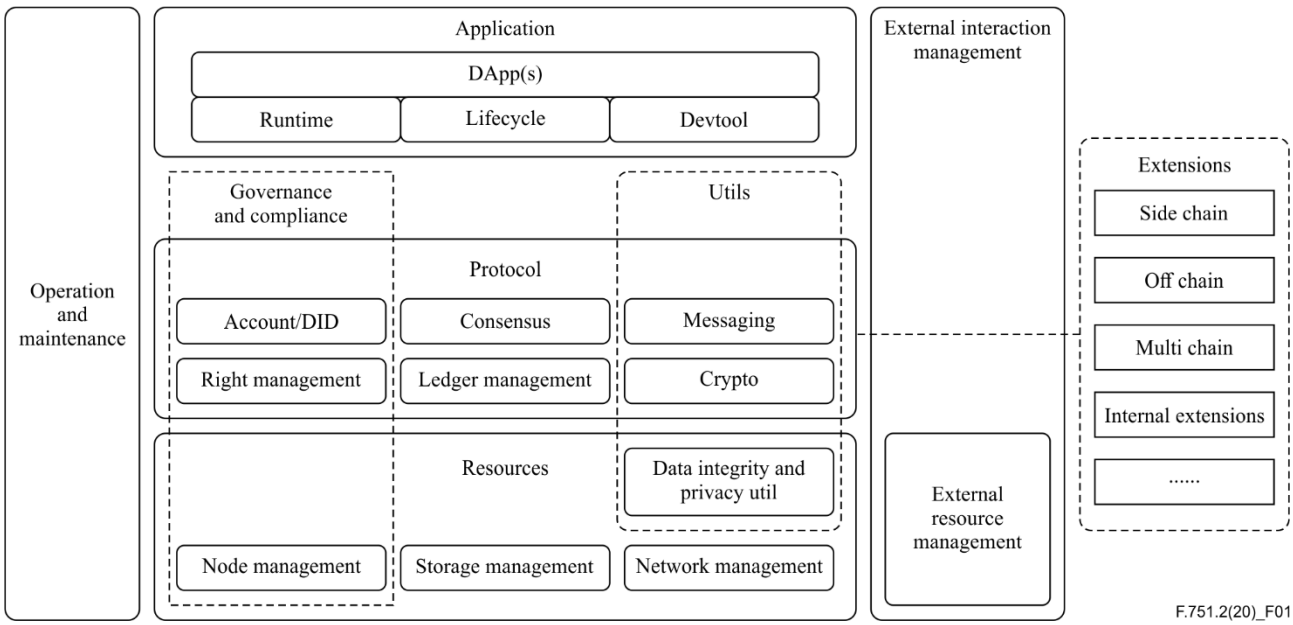


Figure 1 – High-level conceptual architecture of DLT

6.2 Resource and infrastructure functions

The infrastructure provides the operating environment and basic components required for the normal operation of the distributed ledger system. The base layer includes network management functions, storage management functions, utility functions and node management functions. This layer is the resource that most software systems rely on and is the foundational support of the distributed ledger system.

– Network management functions – Each DLT system is built upon a network hypothesis, which leads to the distribution model of the system. For example, in the study of bitcoin, each node inside bitcoin network has the same privileges, thus a peer-to-peer (P2P) network model is used.

– Storage management functions – Each DLT system has a standard storage component to persist data and ensure data integrity and privacy. In particular, based on the cost of distributed storage, storage management may need to provide solutions for on-chain business to balance cost and data protection.

– Utility functions – DLT has utility functions to protect data integrity – not only raw data, but also a runtime channel for data transfer.

– Node management functions – Each node inside a DLT system is maintained by a node owner or operators. Node management is a component to manage the resource of a single node inside a DLT system.

6.3 Protocol or governance and compliance functions

In DLT systems, blockchain systems in particular, each node can have its own implementations based on the system's technical specifications. The protocol layer is a conceptual layer to serve the technical specification across nodes inside a DLT system.

The protocol layer includes: governance and compliance; consensus; ledger management; and utility. Furthermore, the governance and compliance includes: node management; and authentication, authorization and accounting (AAA) management; (account management and right management). The governance and compliance functions are to support the management of system governance (based on trust endorsement hypothesis) and AAA functions of other components. The utility includes: messaging functions; and cryptographic functions. The utility functions are used to support data privacy and integrity protection.

6.3.1 Consensus mechanism functions

A consensus mechanism is the core component of a distributed ledger, especially a decentralized one, and is used to ensure the consensus of all nodes on the data. The consensus mechanism contains consensus algorithms, data validation, data distribution and synchronization. By use of the consensus mechanism, the distributed ledger system sets up a trust mechanism upon the network hypothesis. Any trust endorsement module, e.g., incentive mechanisms, is built upon that. Consensus mechanisms can maintain public data inside a chain and data based on various distributed ledger-partitioning mechanisms.

6.3.2 Ledger management functions

The ledger provides basic data management of distributed ledgers and distribution management of ledger data on the network. It determines the local data storage methods of ledgers and the synchronization mechanism between nodes, and responds to rights management by use of the consensus mechanism.

6.4 Application functions

Based on the runtime management, decentralized applications (DApps) are built to serve different business requirements in a distributed network environment.

Furthermore, web applications with a combination of both off- and on-chain services can be a solution for businesses.

6.4.1 Smart contract mechanism functions

Most DApps are written by smart contracts.

Smart contracts are programs written on the distributed ledger system that encode the rules for specific types of distributed ledger system transactions in a way that can be validated and triggered by specific conditions.

The smart contract mechanism includes language specification, compilation and execution of the code. Smart contracts for different DLT systems can be implemented using simple interpreted scripts or programming languages.

The smart contract mechanism is an optional, but useful, component for DLT.

6.4.2 DApp management functions

DApp management functions provide a tool set for DApp development, manage the runtime of DApp and control the lifecycle of DApp.

An open DApp management layer is middleware that connects the DLT network with the DApp business.

The DApp management middleware contains a DApp framework for DApp developers to create and maintain DApps, and a smart contract management mechanism for DApp hosts to manage their DApps easily.

Furthermore, the DApp framework can provide a series of interfaces for DApp developers. The interface enables the use of distributed ledgers. DApp users access DLT services through the interface. The interface can usually have an application programming interface (API), software development kit (SDK), remote procedure call (RPC), and so on.

6.5 Operation and maintenance functions

Operation and maintenance functions include various libraries, such as log, monitoring, node or network management and scaling libraries.

6.6 External interaction management functions

Each DLT system has its own network hypothesis, trust endorsement hypothesis and governance model. Thus, a DLT system with an open-network hypothesis is able to interact or interoperate with external system(s).

The network hypothesis describes the node scalability of the business, e.g., whether a network is open and node counts.

Trust endorsement judges whether a chain is permissioned or permissionless.

The governance model guides the management of the behaviour of individual nodes.

These three components determine whether the DLT system is able to support external interactions.

In most cases, external interaction management is the runtime engine of external resource management.

6.7 Extension functions

The extension component of a DLT platform aims to resolve different requirements of data interoperability. Extension functions include a series of protocols or specifications for data interoperations of external systems, e.g., multi-chain, side-chain, off-chain, or internal systems, e.g., child-chain, sharding[1] [b-shard1], [b-shard2].

6.7.1 Internal system extensions

Each DLT system has one governance model. The internal system extension aims to resolve the problem of scalability for one DTL ecosystem with the same governance model.

Child chain

Child chains are individual ledgers with their own native tokens responsible for operational transactions, such as deploying smart contracts, issuing assets, voting on polls and sending messages. All child chains receive consensus from, and share the same source code as, the network's main (parent) chain; therefore, all child chains on the network are interoperable.

6.7.2 External system extensions

In most business cases, the data interoperability is a cross-system requirement. A DLT system can access external systems to satisfy business requirements.

Off-chain system(s) functions

In the study of DLT, interaction or interoperation with off-chain systems are scenarios in most use cases.

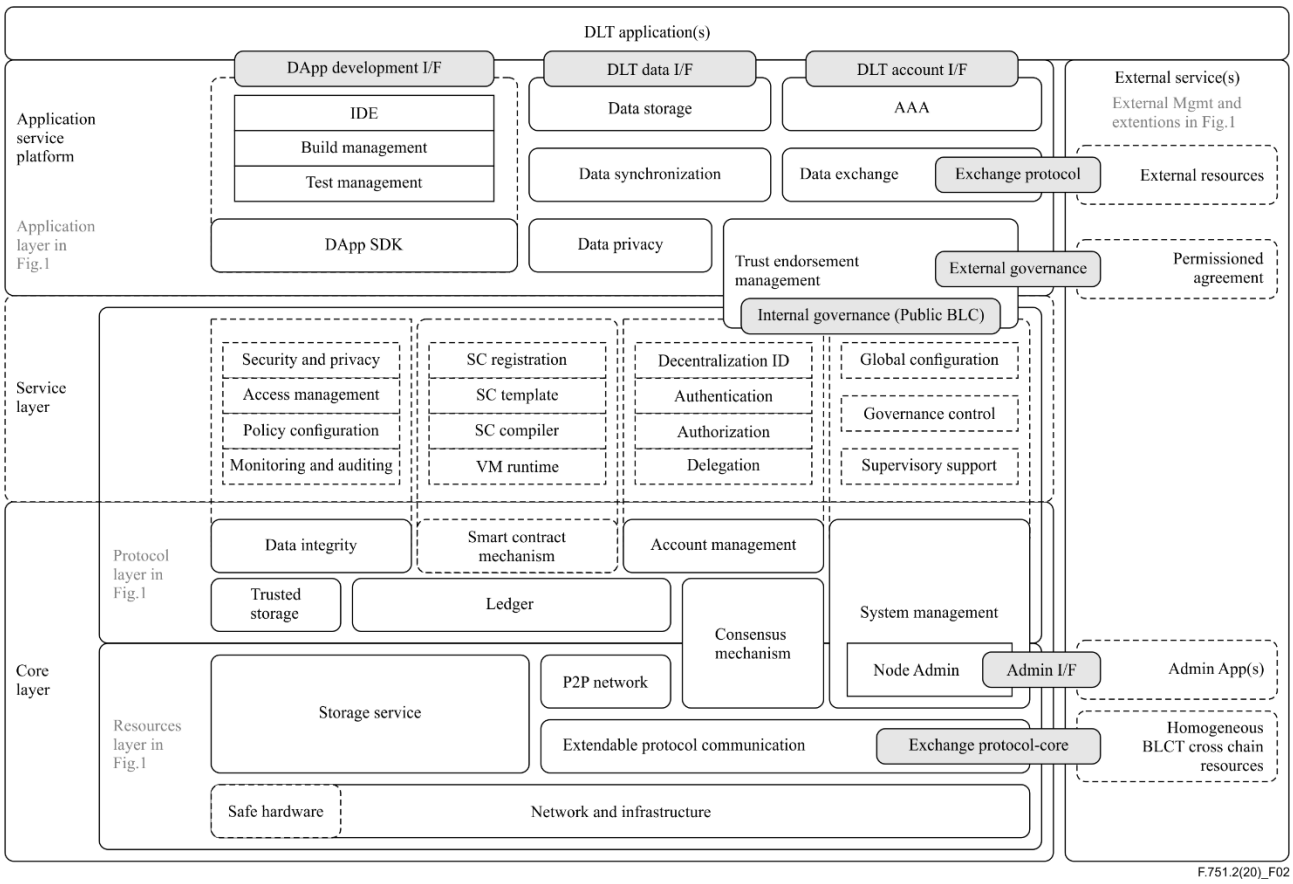
Furthermore, a series of layer 2 (L2) solutions, a combination of on-chain and off-chain techniques, are used to optimize the performance, as well as the scalability, of DLT systems.

7 Functional components

The different distributed ledger platforms are highly consistent on the top-level architecture, but the components in the detailed architecture are different. Clauses 7.1 to 7.5 explain in detail the components and functions of the detailed architecture, which varies slightly from the top-level architecture described in clause 6.1. See Figure 2.

---

[1] Sharding has many solutions, some share the same chain protocol, some have their own extensions, e.g., message queue, side-chain with the same governance model.



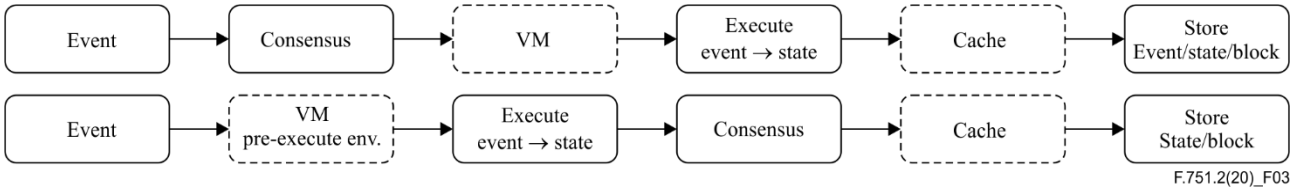
NOTE – Dashed borders indicate optional components.

Figure 2 – Schematic diagram of the detailed architecture

7.1 Core layer

Functions map to resources, protocol and operation and maintenance in Figure 1.

The DLT system aims to execute a transaction; however, it receives events that will not always result in transactions. Figure 3 presents two options of how DLT systems process events of incoming transactions and store the result in the ledger.



NOTE – Dashed borders indicate optional components.

Figure 3 – Two options for typical flow of DLT systems

The standard process involves:

– event – to gather event(s) from client(s);

– virtual machine (VM) – to prepare the environment for event execution;

– execute – to execute the transaction(s) inside the event producing the state result – the state result stands for the status of the ledger;

– store – to store data (state or event).

In decentralized systems, a consensus mechanism is required to solve data consistency between different nodes.

There are two modes for approaching this in DLT systems:

– state mode: consensus upon states, mostly used for pre-execution, UTXO models.

– event mode: consensus upon events, mostly used for post execution, balance models.

7.1.1 Network and infrastructure

As an IT solution, DLT nodes use the same network solutions of typical distributed system solutions (including cloud solutions).

7.1.1.1 Safe hardware

One option to enable DLT systems to support data privacy enhancement with high performance is to use trusted or /safe hardware, e.g., a trust execution environment (TEE).

7.1.2 Extendable protocol communication

Scalable protocol communication module based on network hypothesis across distributed systems.

The communication component aims to give the capability of data exchange across different chain networks with L7-filtering[2] for a homogeneous DLT network, if possible.

7.1.3 Network (P2P network) management

A DLT system is based on a network hypothesis, thus network management is required. Especially in permissionless DLT systems, each node has the same privileges, the node owner is able to decide the contribution of the node itself, network management is to provide basic capability to control nodes in the network.

7.1.3.1 Network discovery

In a distributed ledger system, there are usually many nodes, especially in public permissionless DLT systems. Each node needs to discover neighbour nodes through a network discovery protocol and establish a link with neighbour nodes. According to different permissioned distributed ledger architectures, the network discovery protocol also needs to identify and authenticate the node identity to prevent attacks such as a Sybil attack.

7.1.3.2 Data transceiver

After a node connects to a neighbour node through the network discovery protocol, data exchange (e.g., transaction broadcast, consensus message and data synchronization) with other nodes is completed by the data transceiver module. According to the architecture of different distributed ledgers, the design of the data transceiver needs to consider requirements, e-g., confirmation and encryption.

7.1.4 Storage service

7.1.4.1 Data persistence

Distributed ledgers need to persistently store a variety of data, such as block data, transaction data, status data and private data of a local account. Depending on the type of data and the design of the distributed ledger, different storage modes can be used. Storage modes include relational databases such as MySQL [b-mysql], non-relational databases such as LevelDB [b-leveldb] and self-organizing files.

---

[2] ISO application layer data filter.

7.1.5 Consensus mechanism

7.1.5.1 Data synchronization

The data synchronization module ensures that the distributed ledger has a consistent ledger. The data synchronization module transmits the new part of the ledger between different nodes. The synchronization module also validates the synchronized data to ensure the correctness and consistency of the synchronized data.

Different types of nodes can have different synchronization methods, which include synchronizing all transactions, blocks and status data, synchronizing all transactions and block data and synchronizing partial transaction data.

7.1.5.2 Legitimate validation

Consensus is the core of distributed ledgers. The purpose of consensus is to make many nodes involved in accounting jointly maintain the same ledger. It should be noted the validation refers only to algorithms that agree on the transaction order, and not verification of the transaction data itself.

Consensus algorithm design should ensure the following:

– consistency: eventual consensus node agreement on the data;

– timeliness: completion by consensus nodes of data consensus in as short a time as possible;

– security: a substantial cost barrier to the undermining of consistency and to provide protection against attack.

In theory, all kinds of algorithms can meet the requirements in the previous paragraph and be applied in a given setting as the consensus algorithm of a distributed ledger.

7.1.5.3 Verification

The module contains a transaction buffer pool for distributed ledgers and basic verification of transactions. Distributed ledgers typically verify the sender's balance of the transaction, the sender's transaction number, etc. Once the transaction is validated, the transaction is saved in the DLT system. The verification module can effectively prevent various attacks and ensure the stable operation of the distributed ledger.

7.1.5.4 Consensus algorithm

Proof of work (PoW) [b-pow], commonly known as mining, can be simply understood as providing proof that a certain amount of work has been done. PoW requires the node to carry out a certain amount of computation to obtain the accounting right, which means that it takes a certain amount of computation resource, which results in energy consumption by the computer and calculates a verifiable result through a mathematical operation. The node sends the data that needs to be recorded in this round. After verification, the other nodes in the whole network store the data.

The main feature of the PoW system is that the prover needs to do more work to get the result, while the verifier can easily verify whether the prover has done the corresponding work through the result. A core feature of this approach is asymmetry: work is more difficult for a prover and easier for approver (such as hash algorithm).

Series of Byzantine fault tolerance algorithm. Byzantine fault tolerance (BFT) is a common solution to achieve efficient fault tolerance. The system comes from the Byzantine general problem. A system based on the BFT algorithm can reach valid consensus when the number of failed or fraudulent nodes is less than the number of fault-tolerant nodes. Generally speaking, the number of fault-tolerant nodes is less than one-third of the total number of nodes.

At present, there are many implementations of BFT series algorithms. The typical implementation method is the state machine copy replica algorithm. Under the control of the master node, all nodes confirm the status of other participants through the three-phase protocol and determine the accounting

according to the status of all participants' content. BFT series algorithms currently include the synchronized PBFT algorithm [b-pbft], asynchronous Honey Badger BFT [b-hbft] algorithm and open channel BFT algorithm. There are also some BFT algorithms that support dynamic changes in the number of nodes.

Proof of stake (PoS). PoS allows so-called token holders to replace the miners. The accountant is the holder of relevant tokens and the accounting right is their stake. A typical PoS reduces the difficulty of mining in equal proportion to the percentage and time tokens that each node occupies in order to find a verifiable result and determine the ownership of accounting rights faster.

Delegated proof of stake (DPoS) [b-dpos]. Each token holder determines the accounting right of a DLT system by voting, similar to the election of a board of directors. All nodes whose votes exceed the agreed votes become system trustees, forming a so-called board of directors and alternately signing blocks. There are incentives for encouraging directors never to miss signing a block, but if it occurs, other nodes on the network may vote for a new director to take their place.

Traditional consensus algorithms. Traditional consensus algorithms are based on traditional distributed consensus techniques, including Paxos [b-Lamport], Raft [b-raft], and others. There are many similarities between the traditional and other consensus algorithms, all of which aim to solve data inconsistency caused by failure of network or hardware and data loss in the system. The traditional consistency algorithm pays more attention to performance and has higher requirements on the environment. At present, the traditional consistency algorithm is also widely used in DLT systems.

Hybrid consensus algorithm: The current consensus algorithms have their strengths and weaknesses, so there are some cases where consensus algorithms are mixed, including:

BFT-RAFT [b-Clow], BFT combined with the high performance of Raft and support for fraud nodes;

delegated Byzantine fault tolerance (dBFT) [b-dbft], authorized BFT algorithm, integrated BFT with election and authorization mechanism for more node scenes;

Byzantine fault tolerance with verifiable randomness (VBFT) [b-vbft], achieves chain scalability by consensus node selection with virtual routing and forwarding, anti-attack ability by randomness and PoS, and fast state finality with BFT.

7.1.6 Smart contract mechanism

Smart contracts are programs written on the distributed ledger system, which encode the rules for specific types of distributed ledger system transactions in a way that can be validated and triggered by specific conditions.

The smart contract mechanism handles trusted data processing, including smart contract lifecycle management, contract registration, pre-authorization, deployment, upgrading, iteration and cancellation.

The smart contract mechanism usually includes the contract execution engine, contract code management and contract data management.

Contract code management is responsible for the operation of the availability, deployment and storage of the contract code.

Contract data management maintains contract data and provides an access interface for the contract engine.

The contract execution engine can execute the contract code and maintain the contract data according to the contract code.

7.1.6.1 Language and compiler

For very limited business requirements, you can use a script to implement a contract, which is fast, secure and low in resource consumption.

For lightweight businesses, an inline contract engine is recommended that is highly efficient and reduces the complexity of deployment.

For heavyweight businesses, an external execution environment is recommended that has more resources and higher execution efficiency. These are generally compatible with mainstream programming languages and are easy to develop.

Language and compiler provide the grammar specification of the contract, as well as the compilation specification to ensure that the transaction can be processed by the execution engine.

7.1.6.2 Execution engine

External contract execution environment: It uses the external execution environment to execute the contract code, and usually uses an external container (such as Docker [b-dock]) or virtual machine to ensure a consistent execution environment for all nodes. The maintenance of contract data is accomplished by the distributed ledger process communicating with a process in the external environment, a typical representative includes the realization of the contract in the Docker container of Hyperledger Fabric.

The advantages of this contractual technology are that they can be Turing complete, utilize easy-to-master programming languages and offer efficient contract development. The heavyweight business is more efficient due to the permanent contractual execution environment. The disadvantages are slow deployment of contract code, weak external engine attack resistance, large resource usage, large inter-process communication overhead and unsuitability for short contract codes.

Inline contract engine: A contract engine is embedded in the process, and the contract codes are interpreted during execution or implemented after being compiled. The engine can use open third-party engines, such as the Java virtual machine (JVM) [b-jvm], LUA [b-lua], JS [b-js] and other scripts, or be implemented by the DLT platform, such as the Ethereum virtual machine (EVM) [b-evm]. At present, most distributed ledgers use this technology, e.g., Ethereum.

The advantages of inline contract engines include that they can be Turing complete, offer fast contract deployment, speed up data access, harden security of the engine, and provide convenient management and customization of the computing resources of the contract engine. The disadvantages are that the contract is generally not resident in memory, and the engine efficiency is difficult to compare with the external engine, making it less efficient to perform heavyweight business.

Scripting or finite state machine: A set of execution instructions is pre-set in the distributed ledger, this sequence forms a piece of contract code, which is interpreted by the state machine. Due to the complexity of the instructions and the size of the code, the script generally has a weak function and is not Turing complete, meaning it only performs basic operations. Many current virtual currency ledgers, including Bitcoin, use this technology.

Advantages include that scripting technology is easy to implement, its script functions are effectively controlled and there is strong anti-attack mitigation. Disadvantages include complex script development, Turing incompleteness, and the scale of scripts is small, resulting in limited functions.

7.1.7 Ledger

The ledger stores all the transaction data and contract data. The ledger needs to be able to complete the transaction and the processing,[3] indexing, and storage of contracts.

Ledger management contains two components:

– ledger mechanism – based on consensus mechanism, relying on P2P networking and extended storage services;

– ledger storage – extended storage services for ledger, e.g., databases.

---

[3] For those transactions that do not involve smart contract.

Consensus algorithm design should ensure the following:

– there are complete definitions of the various data types in the ledger, such as transactions, blockchain, assets and accounts;
– the transaction data can be summed up using algorithms, such as the Merkle tree, the blocks can be continuously added, and successive blocks are chain linked to ensure the integrity of the consensus;
– the design of the ledger can be used for data synchronization and verification, and the ledgers should support large-scale data storage and high concurrency.

7.1.7.1 Transaction record

Account: The account employs asymmetric cryptography to create a public and private key pair that controls access to the account, which can authenticate account ownership.

Assets: Assets can be expressed in the form of account and balance models, or in the form of a transaction output (TXO; e.g., UTXO [b-utxo]) model, or a combination of the two. Assets can also be specified in the contract data independent of the underlying assets on the ledger.

Ledger storage: The ledger storage is usually implemented as a custom file or as an embedded database. The ledger store is divided into two parts: one part is the transaction history, usually the block data, which contains the original transaction; the other part is the result of the transaction or contract execution, usually stored outside the block.

Block structure: A block consists of the transactions, a block header, a variety of relevant data, and its own block digest. The digest of the block is usually calculated from the transactions in the block, the digest of the previous block, and various other related data according to an algorithm such as the Merkle tree. It is a type of data structure to keep transactions in a DLT.

7.1.7.2 Status

Most distributed ledgers, in addition to saving blocks and transaction data, also hold some data or results of the transaction execution, which can be called status. The status of different nodes of the distributed ledger should be consistent. For a simple distributed ledger like Bitcoin, the status may contain only a list of unspent assets, and so on. For distributed ledgers like Ethereum and Hyperledger Fabric with Turing-complete intelligent contract engines, the status preserves more complex data, generated during virtual machine execution.

By maintaining status data, distributed ledgers can continue to process a series of complex and continuous transactions.

7.1.8 Data integrity and trusted storage

DLT platforms should provide data integrity capabilities for data reading from and writing to ledgers. They should use trusted storage to ensure that data is tamperproof.

7.1.9 Account management

Account management manages entities, including users and nodes, and data identifiers in a DLT system, which can be expanded to different modules for different services.

7.1.10 System management

System management manages nodes and the DLT network, which can be expanded to different modules for different services.

Node management of DLT system management provides an operation and maintenance feature, including node deployment and node administration.

The deployment of distributed ledger refers to the installation and use of distributed ledger services for different scenarios and users, with different node permissions and service modes. According to

the type, distributed ledgers are divided into permissionless and permissioned DLTs, where their deployment methods are not the same.

Permissionless DLT

Permissionless DLTs generally do not make any restrictions on node access and are less demanding on the operating environment. The ledger nodes are relatively simple. All nodes are free to participate in consensus and read and write data.

Permissioned DLT

In such distributed ledgers, consensus processes can only be involved with authorized customer nodes. Authorized nodes can participate in the consensus and data read and write process according to the rules.

The deployment of DLTs can occur in the following ways.

– Process-based deployment – Operations staff personnel deploy nodes on the host or virtual machine to complete the configuration. This deployment is flexible. However, due to the complexity of the configuration file, the deployment is inefficient and prone to problems caused by inconsistent node environments.

– Container-based deployment – This involves ensuring a uniform environment within the container and encapsulating the distributed ledger in the container, then deploying the container by the operations staff. This approach is simpler to deploy than process-based deployment and is more reliable.

– Cloud service-based deployment – Integration of distributed ledger and cloud services that facilitates rapid deployment of distributed ledger services through the cloud platform, while providing different levels of services, such as platform as a service, backend as a service.

– Hybrid-based deployment –An approach that can combine cloud service-based deployment with any of the two other ways mentioned here to address cloud perspectives and possible on-premises compliance.

7.1.11 Utility

Core layer components share the same technical utility modules. Utility functions map to utility in Figure 1.

7.1.11.1 Cryptographic library

Distributed ledgers use a variety of cryptographic algorithms. The cryptographic algorithm library provides basic cryptographic algorithm support for each component, including various commonly used encoding algorithms, such as hash algorithms, signature algorithms and privacy protection algorithms. The cryptographic algorithm library also provides functions such as maintenance and storage of secret keys.

7.1.11.2 Messaging

The message module provides message notification services between different components within the distributed ledger and between different nodes. For example, after a successful transaction, the customer usually needs to track the results of the execution of the transaction and even some records during the execution of the transaction. The message module can complete the generation, distribution, storage and other functions of the message to meet the needs of the distributed ledger.

7.2 Service layer

Functions map to protocol and governance in Figure 1.

The service layer is middleware between the core and application service platform layers. Their services are an extension from four basic services in the core layer; data integrity; smart contract mechanism; account management; and system management.

The services described in the service layer are optional.

7.2.1 Expandable services for account management

A distributed account system serves all kinds of entities and data in a DLT system. The account management component controls addresses and identities.

– Address is bound to trust data. Especially in public chain projects, a large part of trust data is usually digital asset.

– Identity is bound to entity.

Based on decentralized identity (DID) and multi-dimensional authentication protocol, the account management component provides the identity management of node operators, trust endorsement regulatory roles, DApp operators, end users and extended Internet of things access entities.

The account management of identity usually utilizes the following technologies:

– certificate authority (CA): the certificates are issued through a centralized CA for various applications in the system, and both the identity and authority management are certified and confirmed by these certificates;

– identity-based encryption **(**IBE**)**: the identities are confirmed by IBE;

– public key infrastructure **(**PKI**)**: the identities are confirmed by addresses or accounts based on PKI;

– third party identity authentication: the identities are confirmed by the third party.

7.2.1.1 Decentralized identity

DID establishes a cryptographic-based digital identity for entities (nodes, users, data, things, etc.) in a DLT system. The digital identity is based on DLT and is not subject to any centralized organization. It is potentially controlled by multiple entities and is both secure and trustworthy.

7.2.1.2 Authentication and authorization

AAA solutions support the full use of account management.

Approaches to authority management differ since a variety of distributed ledgers have different application scenarios.

The existing business expansions can interface with the existing authentication and authority management.

The more concentrated authority management business can be complemented by using CA/IBE.

For instance, the public chains usually adapt PKI technology rather than central authority management like CA/IBE.

7.2.1.3 Delegation

In DLT account management, non-user entities can also use distributed identities. The use of non-user entities shall enable ID delegation, where entity owners can make full use of them.

7.2.2 Expandable services for system management

System management includes secure communication, trusted data transmission, and related services of node operation and network governance.

7.2.2.1 Global configuration

Global configuration for nodes inside a DLT system includes network configuration, communication management configuration, node system configuration, etc.

7.2.2.2 Governance control

Governance control is required for individual nodes and a DLT system via multiple nodes, including network status, communication channel monitoring, alarm and tracking, trust endorsement and node failure monitoring.

7.2.2.3 Supervisory support

Under this heading comes component support for the monitoring and supervision of malicious events in a DLT system.

7.2.3 Expandable services for smart contract mechanism

In the service layer, the smart contract mechanism provides the component for contract registration, contract template, contract compiler and VM runtime.

7.2.4 Expandable services for data integrity

7.2.4.1 Policy configuration

The policy configuration of trusted data includes access templates, privacy templates, and monitoring and auditing strategies.

7.2.4.2 Access control

Under this heading comes trusted data access control and dynamic data operation control.

7.2.4.3 Data security and privacy management

Under this heading comes security management of data storage and privacy management for data affirmative easement.

7.2.4.4 Data monitoring and auditing

Tracking and monitoring of data, data processing, review and audit of special data events.

7.2.5 Trust endorsement management

According to the different trust endorsement approaches of distributed systems, trust endorsement management is used to meet the governance model, e.g., the legal endorsement of off-chain governance in chains that unite nodes with agreements or contracts; the tokenomics endorsement of on-chain governance in public chains.

7.2.5.1 Incentive module

The incentive module is used for accounting incentives for partially distributed ledgers, such as typical public chains. Most public chains use consensus algorithms for targeted token distribution and for motivating the accounting nodes to account. In general, the incentive module is very closely related to the consensus algorithm and governance model.

7.3 Application service platform

Functions map to application in Figure 1.

7.3.1 DApp service interfaces

DApp services include interfaces that support DApp development, DLT data management and DLT account management.

The upper layer interface provided by the distributed ledger gives external systems efficient access to distributed ledger data, to external applications to integrate distributed ledgers or to other distributed ledgers for mutual access, usually including RPC, API and SDK. The interface layer mainly completes data synchronization, processing, transferring and exchange, etc.

The RPC interface connects peripheral components with the distributed ledger nodes over the network and to access the services provided by the distributed ledger. SDK provides a development package for other components to integrate part of the functionality of a distributed ledger.

RPC and SDK should observe the following rules:

– completely functional: the transactions of distributed ledger can be completed and maintained, and an intervention strategy and privilege management are operational;

– portable: it can be used in a variety of applications and environment, and is not limited to one absolute software or hardware platform;

– extensible and compatible: it should be as forward and backward compatible as possible, and try not to modify or minimize changes as extending functions;

– easy to use: the structured design and good naming methods should be used to reduce the cost of development.

Common implementation techniques include call control, serialized objects, and network components. There are various architectures that can be used, such as CORBA [b-corb], JsonRPC [b-jrpc], gRPC [b-grpc], Thrift [b-thrft], RestAPI [b-rapi] and XMLRPC [b-xrpc].

7.3.2 Accounting, authorization and authentication

The AAA system manages the DApp user's ability to access data, process data and perform data exchange based on transactions.

The AAA management shall focus on three parts:

– access control for DApp users to submit their transaction;

– access control for nodes to access the chain network (permission control), usually being applied by a permissioned chain;

– privilege control for DApp users to operate the data and function calls via transactions.

7.3.3 Data privacy

Cryptographic techniques are used to protect on-chain user data and to meet data privacy requirements for applications (result in DApps).

Privacy has always been one obstacle to the application of distributed ledgers. A successful means of satisfying regulatory requirements while not infringing data privacy is key to the distributed ledger industry.

Privacy protection should therefore meet the following requirements:

– anonymity controls: to ensure that users can set the transaction, so it is not visible to unrelated parties;

– high performance: privacy-protected design must still be able to meet performance requirements;

– transparent supervision: to satisfy the requirement that privacy protection should not evade the regulatory functions of regulatory agencies – technically, this feature is optional.

7.3.4 Data storage and synchronization

Useful tools for end users and DApps to facilitate DApp data storage, synchronization, operation and credentials.

7.4 DLT applications

Functions map to application in Figure 1.

Multiple applications based on DLT systems are called DApps. DApps are not part of DLT systems, they are consumers instead.

7.5 DLT extendable interfaces and external services

Functions map to external interaction management and extensions in Figure 1.

The reference architecture of DLT, especially a blockchain system, shall meet the requirement to balance the needs of security, decentralization and scalability. Furthermore, decentralized systems focus on resolving trust issues in competitive business environments, therefore, not all business cases shall use decentralized systems.

A hybrid system combines DLT with typical IT systems and can satisfy most business requirements.

From a DLT perspective, external services provide solutions to cooperate with external systems.

7.5.1 Administration interfaces

Administration interfaces provide capability for external systems to create applications to communicate with DLT services and manage DLT nodes if possible.

7.5.2 External governance interfaces

For DLT systems that use trust endorsement for nodes externally, they should provide external governance interfaces, e.g., permissioned agreements or contracts signed by nodes.

7.5.3 Exchange protocol and extensional resources

Exchange protocols define the way DLT system interact with external resources in non-DLT systems, third party DLT systems or L2 blockchain technology.

Many extensions are provided to implement exchange protocols.

The main purpose of L2 blockchain technology is to scale blockchain transaction capacity while retaining the benefits decentralization brings to a distributed protocol. Solving the scalability problem will significantly help with blockchain's mainstream adoption. Layer 2 blockchain technology systems are those that connect to and rely on blockchain systems as a base layer of security and finality.

L2 solutions include plasma, state channel, sharding, raiden network and lightening network.

External interaction or interoperation management

In L2 solutions, the DLT system is able to interact or interoperate with L2 systems.

External resource management

To cooperate with non-DLT systems and third party DLT systems, mostly data or resource exchange transactions, resource management is required.

Bibliography

[b-ITU−T F.751.0] Recommendation ITU-T F.751.0 (2020), *Requirements for distributed ledger systems*.

[b-ITU-T TS FG DLT D3.1] Technical Specification ITU-T FG DLT D3.1 (2019), *Distributed ledger technology reference architecture*.

[b-ISO 22739] ISO 22739:2020, *Blockchain and distributed ledger technologies – Vocabulary*.

[b-Clow] Clow, J., Jiang, Z. (2017). *A Byzantine fault tolerant raft.* Stanford, CA: Stanford University. 6 pp. Available [viewed 2020-10-14] at: http://www.scs.stanford.edu/17au-cs244b/labs/projects/clow_jiang.pdf

[b-Lamport] Lamport, L. (1998). The part-time parliament. *ACM T. Comput. Syst.*, **16(**2), pp. 133-169. Updated version available [viewed 2020-10-14] at: http://lamport.azurewebsites.net/pubs/lamport-paxos.pdf

[b-bitc] Bitcoin (Internet). *Bitcoin is an innovative payment network and a new kind of money.* Available [viewed 2020-10-14] at: https://bitcoin.org/en/

[b-corb] CORBA (Internet). *Common Object Request Broker Architecture*. Milford, MA: CORBA. Available [viewed 2020-10-14] at: http://www.corba.org/

[b-dbft] NEO (Internet). *Consensus mechanism.* Available [viewed 2020-10-14] at: https://docs.neo.org/docs/en-us/basic/technology/dbft.html

[b-dock] Docker (Internet). *Get started with Docker*. Palo Alto, CA: Docker Inc. Available [viewed 2020-10-14] at: https://www.docker.com/

[b-dpos] Bitshares (Internet). *Delegated proof-of-stake consensus*. Blacksberg, VA: Bitshares. Available [viewed 2020-10-14] at: https://bitshareshub.io/delegated-proof-of-stake-consensus/

[b-ethe] Ethereum (Internet). *Ethereum is a global, open-source platform for decentralized applications*. Zug: Etherium Foundation. Available [viewed 2020-10-14] at: https://www.ethereum.org/

[b-evm] Chinchilla, C., editor (2020). *Ethereum virtual machine (EVM) awesome list*. San Francisco, CA: Github. Available [viewed 2020-10-14] at: https://github.com/ethereum/wiki/wiki/Ethereum-Virtual-Machine-(EVM)-Awesome-List

[b-grpc] Cloud Native Computing Foundation (Internet). *About gRPC*. San Francisco, CA: Linux Foundation. Available [viewed 2020-10-14] at: https://grpc.io/

[b-hbft] Miller, A. (2020) *The honey badger of BFT protocols*. San Francisco, CA: Github. Available [viewed 2020-10-14] at: https://github.com/amiller/HoneyBadgerBFT

[b-jrpc] Morley, M. (Internet). *JSON-RPC*. MPCM Technologies LLC. Available [viewed 2020-10-14] at: https://www.jsonrpc.org/

[b-js] Pluralsight (Internet). *Ready to try JavaScript?*. javascript.com. Available [viewed 2020-10-14] at: https://www.javascript.com/

[b-jvm] Tyson, M. (2018). *What is the JVM? Introducing the Java virtual machine*. Framingham, MA: IDG Communications. Available [viewed 2020-10-14] at: https://www.javaworld.com/article/3272244/core-java/what-is-the-jvm-introducing-the-java-virtual-machine.html

[b-leveldb] Ghemawat, S., Dean, J. (2020). *LevelDB.* San Francisco, CA: Github. Available [viewed 2020-10-14] at: https://github.com/google/leveldb

[b-lua] Ierusalimschy, R., Celes, W., de Figueiredo, L.H. (Internet). *Lua*. Available [viewed 2020-10-14] at: https://www.lua.org/

[b-mysql] Oracle (Internet). *MySQL*. Redwood City, CA: Oracle. Available [viewed 2020-10-14] at: https://www.mysql.com/

[b-raft] Raft (Internet). *The Raft consensus algorithm*. San Francisco, CA: Github. Available [viewed 2020-10-14] at: https://raft.github.io/

[b-rapi] RESTfulAPI.net (2020). *What is REST?* Available [viewed 2020-10-14] at: https://restfulapi.net/

[b-shard1] Ethereum (2020). *Sharding FAQs.* San Francisco, CA: Github. Available [viewed 2020-10-14] at: https://github.com/ethereum/wiki/wiki/Sharding-FAQs

[b-shard2] Ontology (2020). *Ontology Sharding PDF*. San Francisco, CA: Github. Available [viewed 2020-10-14] at: https://github.com/ontio/documentation/blob/master/sharding/ontology-sharding.pdf

[b-thrft] Apache Software Foundation (2017). *Apache Thrift*. Wakefield, MA: Apache Software Foundation. Available [viewed 2020-10-14] at: http://thrift.apache.org/

[b-utxo] Bitcoin Project (2009-2020). Unspent transaction output (UTXO). In: *Transactions*. Arlington, VA: Bitcoin. Available [viewed 2020-10-14] at: https://developer.bitcoin.org/devguide/transactions.html

[b-vbft] Ontology (2020). *VBFT introduction*. San Francisco, CA: Github. Available [viewed 2020-10-14] at: https://ontio.github.io/documentation/vbft_intro_en.html

[b-xrpc] XML-RPC (Internet). *What is XML-RPC?* San Francisco, CA: Github. Available [viewed 2020-10-14] at: http://xmlrpc.scripting.com/

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series D	Tariff and accounting principles and international telecommunication/ICT economic and policy issues
Series E	Overall network operation, telephone service, service operation and human factors
Series F	**Non-telephone telecommunication services**
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Environment and ICTs, climate change, e-waste, energy efficiency; construction, installation and protection of cables and other elements of outside plant
Series M	Telecommunication management, including TMN and network maintenance
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling, and associated measurements and tests
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks, open system communications and security
Series Y	Global information infrastructure, Internet protocol aspects, next-generation networks, Internet of Things and smart cities
Series Z	Languages and general software aspects for telecommunication systems

Printed in Switzerland
Geneva, 2020